ATLAS

MANUAL DE USUARIO

Servicio de Invocación De Servicios Para Axis 2

Versión 1.0

Área de Integración y Arquitectura de Aplicaciones



Agencia de Informática y Comunicaciones de la Comunidad de Madrid



Hoja de Control

Título	Manual de Usuario Invocador de Servicios		
Documento de Referencia	NORMATIVA ATLAS		
Responsable	Área de Integración y Arquitectura de Aplicaciones		
Versión	1.0	**Fecha Versión**	16/02/2011

Registro de Cambios

Versión	Causa del Cambio	Responsable del Cambio	Fecha del Cambio
1.0	Versión inicial del documento	Área de Integración y Arquitectura de Aplicaciones	02/11/2010

Índice

1. INTRODUCCIÓN

Muchas de las aplicaciones que se desarrollan para la Comunidad de Madrid necesitan acceder a servicios web tanto servicios que se han desarrollado específicamente para la tramitación electrónica como otros servicios web que incluso pueden estar fuera de nuestros entornos.

Dada la amplia variedad de tipos de servicios web que nos podemos encontrar y los distintos tipos de seguridad que nos pueden requerir los citados servicios web se ha desarrollado el componente Invocador de Servicios de Atlas para facilitar la creación de los clientes de acceso a los servicios y que mediante una sencilla configuración se le pueda incluir los requisitos de seguridad requeridos.

1.1. Audiencia objetivo

Este documento esta orientado a desarrolladores java que quieran invocar a un servicio web desde un aplicativo que se desarrolla con Atlas.

1.2. Conocimientos Previos

Para un completo entendimiento del documento, el lector deberá tener conocimientos previos sobre las siguientes tecnologías:

- Spring Framework.
- Servicios Web
- Axis2 1.5
- WSS4J
- Seguridad (uso básico de certificados, claves y almacenes con openssl y keytool)

2. DESCRIPCIÓN

Este componente se basa en los siguientes elementos:

- Plugin Maven axistools
- Localizador de servicios de Spring
- Axis2 1.5

Se ha creado una extensión del plugin wsdl2code que se llama atlasfrm-clientews-wsdl2code-maven-plugin al cual se le han añadido las siguientes funcionalidades:

- Generación de Test Unitarios
- Generación de la configuración de Spring necesaria para la ejecución de los test

El plugin de Maven se configura indicando cual es el fichero wsdl a partir del cual se va a generar el cliente. A partir de esta configuración mediante Maven se generarán automáticamente las clases clientes y se incluirán en el proyecto. Además se generaran clases de prueba de estos clientes.

En ocasiones nos podemos encontrar con servicios web a los que se les ha dotado de medidas de seguridad. En este documento se explicará más adelante como implementar la comunicación con este tipo de servicios.

3. INSTALACIÓN Y CONFIGURACIÓN

En los siguientes pasos comentaremos como configurar, generar y utilizar el componente invocador de servicios web.

3.1. Inclusión de dependencia en fichero pom.xml

ATENCION
Si se ha partido de uno de los arquetipos del framework Atlas, no es necesario incluir la dependencia.

Añadir en la siguiente dependencia en el fichero pom.xml:

pom.xml

```
<dependency>
    <groupId>atlasfrm</groupId>
    <artifactId>atlasfrm-dep-service</artifactId>
    <version>${atlasfrm-dep-service.version}</version>
    <type>pom</type>
</dependency>
```

3.2. Instalación y configuración del plugin de Maven atlasfrm-clientews-wsdl2code-maven-plugin

Dentro de la sección de plugins del fichero pom.xml de nuestro proyecto es necesario incluir el plugin de Maven atlasfrm-clientews-wsdl2code-maven-plugin, que es una extensión del plugin axis2-wsdl2code-maven-plugin.

Este plugin generará tanto las clases de cliente del webservice como las clases de tests para las pruebas contra dicho servicio.

3.2.1. Generación de un cliente de servicio web

A continuación se muestra un ejemplo de configuración del plugin:

pom.xml

```
<!--  Plugin para invocador de servicios de negocio.
      Descomentar si se desea utilizar esta caracteristica -->
 <plugin>
    <groupId>atlasfrm</groupId>
    <artifactId>atlasfrm-clientews-wsdl2code-maven-plugin</artifactId>
    <version>${atlasfrm-clientews-wsdl2code-plugin.version}</version>
    <configuration>
       <wsdlFile>src/main/resources/wsdl/xxxx_ws.wsdl</wsdlFile>
       <packageName>xxxx.ws.client</packageName>
       <serviceNameAsPackage>true</serviceNameAsPackage>
       <overWrite>false</overWrite>
    </configuration>
    <executions>
       <execution>
        <goals>
          <goal>wsdl2code</goal>
        </goals>
       </execution>
    </executions>
</plugin>
```

Si se ha partido de uno de los arquetipos de Atlas, el plugin se encuentra comentado en el fichero pom.xml del arquetipo, sólo será necesario descomentarlo.

Según la configuración anterior a partir del fichero **xxxx_ws.wsdl** se generarán las clases cliente del servicio **xxxx_ws** en el directorio **src/main/java** en el paquete **xxxx.ws.client.xxxxservice** (suponiendo que **XxxxService** sea el nombre del servicio). También se generará un fichero **applicationContext-XxxxService.xml** en el directorio **src/main/resources**, para que el cliente esté dado de alta en el contexto de Spring. Puede ser necesario incluir manualmente en los **configLocations** del fichero web.xml este nuevo archivo de definiciones.

Además se generarán los test unitarios del servicio en el directorio **src/test/java**, en el mismo paquete en que se han generado las clases del cliente. El test unitario contendrá el fichero de definiciones de Spring creado para el servicio.

3.2.2. Genearción de múltiples clientes de servicio web

En caso de tener que generar varios clientes de servicio web, la estructura de configuración del plugin es diferente a la mostrada en el apartado anterior. En este caso deberán generarse tantos tags **<execution>** como clientes deban generarse, cada uno con su configuración.

No obstante, se recomienda especificar la configuración comun a todas las ejecuciones a nivel del plugin como se muestra en el ejemplo siguiente:

pom.xml

```
<!--  Plugin para invocador de servicios de negocio.
      Descomentar si se desea utilizar esta caracteristica -->
<plugin>
    <groupId>atlasfrm</groupId>
    <artifactId>atlasfrm-clientews-wsdl2code-maven-plugin</artifactId>
    <version>${atlasfrm-clientews-wsdl2code-plugin.version}</version>
    <configuration>
       <packageName>atlasfrm.samples.ws.client</packageName>
       <serviceNameAsPackage>true</serviceNameAsPackage>
       <overWrite>false</overWrite>
    </configuration>
    <executions>
       <execution>
           <id>WService1</id>
           <configuration>
               <wsdlFile>src/main/resources/wsdl/xxxx_ws.wsdl</wsdlFile>
           </configuration>
           <goals>
               <goal>wsdl2code</goal>
            </goals>
       </execution>
       <execution>
           <id>WService2</id>
            <configuration>
                <wsdlFile>src/main/resources/wsdl/yyyy_ws.wsdl</wsdlFile>
            </configuration>
            <goals>
                <goal>wsdl2code</goal>
            </goals>
       </execution>
    </executions>
</plugin>
```

3.2.3. Parámetros de configuración del plugin

Los parámetros que se pueden configurar en este plugin son, además de los propios del plugin **wsdl2code-maven-plugin (**y que podemos encontrar descrita en la siguiente url: http://ws.apache.org/axis2/tools/1_4/maven-plugins/maven-wsdl2code-plugin.html), los incluidos en la siguiente tabla. Se incluyen también los más relevantes y/o los que han variado con respecto al original (cambia su valor o su valor por defecto):

Propiedad	Descripción	Valor
wsdlFile	Esta etiqueta indica la ruta dentro del proyecto al fichero de descripción del servicio web del que se va a generar el cliente.	Ej: src/main/resources/wsdl/xxxx_ws.wsdl Por defecto: src/main/resources/service.wsdl
packageName	Nombre del paquete base donde se generarán las clases java del cliente de servicio web (en src/main/java)	Ej: xxxx.ws.client
serviceNameAsPackage	Si es 'true' se añadirá el nombre del servicio como paquete al final de packageName. El nombre del servicio se modificará para cumplir la normativa de nomenclatura de paquetes.	Por defecto: true
overWrite	Si es 'false' se comprueba si ya existe el fichero y en caso afirmativo, no se generará de nuevo. Se recomienda siempre este valor para que los ficheros se generen solo una vez y no se pierdan modificaciones manuales en las clases generadas.	Por defecto: false
skipGeneratioin	Si es 'true' el plugin no intentará generar ningún cliente. Se recomienda usar este parámetro cuando se modifiquen los nombres de los ficheros generados, de lo contrario se generarán ficheros nuevos con los nombres originales.	Por defecto: false
outputDirectory	Directorio de fuentes donde se generarán las clases del cliente de servicio web	Por defecto: src/main/java
resourceOutputDirectory	Directorio de recursos donde se generarán los ficheros de spring del cliente.	Por defecto: src/main/resources/conf
testOutputDirectory	Directorio de fuentes donde se generarán los tests unitarios	Por defecto: src/test/java
testResourceOutputDirectory	Directorio de recursos de test donde se generarán los ficheros de configuración de Spring necesarios (si los hubiere)	Por defecto: src/test/resources/conf
syncMode	Modo de conexión al webservice	Por defecto: sync (síncrono)

generateTestcase	Indica si generar o no una clase de test para el cliente de webservice.	Por defecto: true
unpackClasses	Genera las clases en distintos ficheros	Por defecto: true
generateServerSideInterface	Genera el intefaz java del servicio web	Por defecto: true
unwrap	Desempaqueta los parámetros de entrada y salida de los métodos del webservice	Por defecto: true
namespaceToPackages	Lista de Namespaces del fichero WSDL relacionandolos con nombres de paquete para generar en estas ubicaciones los datos asociados a cada Namespace.	Por defecto: todas las clases se generan en el mismo paquete del cliente del servicio web.

Una vez que este plugin está configurado se puede pasar a la generación de las clases clientes del servicio web.

4. GENERACION DEL CLIENTE DEL SERVICIO WEB

La generación a partir del plugin **atlas-wsdl2code-maven-plugin** se basa en el documento wsdl para generar el cliente. A continuación de muestra un ejemplo de generación para el servicio OptiwsService. La configuración del plugin es:

pom.xml

```
<!--  Plugin para invocador de servicios de negocio.
      Descomentar si se desea utilizar esta caracteristica -->
 <plugin>
    <groupId>atlasfrm</groupId>
    <artifactId>atlasfrm-clientews-wsdl2code-maven-plugin</artifactId>
    <version>${atlasfrm-clientews-wsdl2code-plugin.version}</version>
    <configuration>
       <wsdlFile>src/main/resources/wsdl/opti_ws.wsdl</wsdlFile>
       <packageName>atlasfrm.samples.ws.client</packageName>
       <serviceNameAsPackage>true</serviceNameAsPackage>
       <overWrite>false</overWrite>
    </configuration>
    <executions>
       <execution>
        <goals>
          <goal>wsdl2code</goal>
        </goals>
       </execution>
    </executions>
</plugin>
```

A continuación se muestra un ejemplo de los ficheros generados para el servicio OptiwsService:

Clases del cliente generadas en **src/main/java**: • Paquete base: **atlasfrm.samples.ws.client** • Nombre del Servicio: **OptiwsService** • Nombre de servicio como paquete: Si • Interfaz del servicio: **OptiwsService.java** • Implementación del cliente: **OptiwsServiceStub.java** • Mapeo de datos: **CodigoDeclaracion2.java**, **CodigoDeclaracion2Response.java**, **ExtensionMapper.java**.	src/main/java atlasfrm.samples.dao atlasfrm.samples.domain atlasfrm.samples.jsf atlasfrm.samples.services atlasfrm.samples.services.facade atlasfrm.samples.ws.client.optiwsservice CodigoDeclaracion2.java CodigoDeclaracion2Response.java ExtensionMapper.java OptiwsService.java OptiwsServiceStub.java
Configuración de Spring generada en src/main/resources: • Configuración de Spring: **applicationContext-OptiwsService.xml**	src/main/resources conf application.properties applicationContext-dao.xml applicationContext-database.xml applicationContext-general.xml applicationContext-OptiwsService.xml applicationContext-security-hostPolitica.xml applicationContext-services.xml applicationContext-webservices.xml politicaDemo.xml politicaProhibido.xml queries.properties
Clases de test generadas en src/test/java: • Test unitario: **OptiwsServiceTest.java**	src/test/java atlasfrm.samples.ws.client.optiwsservice OptiwsServiceTest.java

Una vez generadas las clases del cliente, el fichero de Spring y el test unitario, habrá que editar este último para proporcionar datos para las llamadas al servicio web. Si no se modifica el fichero de test para añadir estos datos, el test unitario fallará.

En cada método de test en que sea necesario aportar datos para hacer la llamada, se generará un comentario como el siguiente para indicar el sitio donde introducir estos:

TestCase.java

```
    // TODO : Rellenar aqui los valores de xxxxxxxxxxxxx
    // El test unitario no ejecutara correctamente hasta que no se
    // rellenen valores correctos
    //
    //
    //
```

5. USO DEL INVOCADOR DE SERVICIOS

En este apartado se demostrará el uso del cliente generado. Se utilizará como ejemplo el cliente del servicio OptiwsService generado en apartados anteriores.

5.1. Parametrización del endpoint

La primera modificación que habrá que hacer al código generado será parametrizar el endpoint por defecto en el fichero '**environment.properties**'. Para ello, se creará una variable '**ws.endpoint.xxxx**' en este fichero, donde '**xxxx**' es el nombre del servicio en minúsculas (o el nombre del paquete final si se utilizó el parámetro **serviceNameAsPackage** como se recomienda). En el ejemplo, el parámetro se llamará '**ws.endpoint.optiwsservice**'.

environment.properties

```
[...]
# Endpoints de webservice
ws.endpoint.optiwsservice=http://desarrollo.madrid.org/optiws/services/OPTIWSSERVICE
```

ATENCIÓN
Los parámetros '*endpoint'* han de crearse en **TODOS** los ficheros *environment.properties* del proyecto, aunque no se sepan las URLs en los distintos entornos.

A continuación habrá que modificar el fichero de Spring, **applicationContext-xxxxService.xml** generado y cambiar el valor del endpoint por defecto por el parámetro creado.

applicationContext-OptiwsService.xml

```
[...]
    <!-- ============================================
         Definicion del endpoint.
         Puede ser necesario insertar el valor a traves del fichero
         environment.properties.
         ============================================ -->
    <bean id="OptiwsServiceEndpoint" class="java.lang.String">
        <constructor-arg value="${ws.endpoint.optiwsservice}" />
    </bean>
[...]
```

5.2. Incluir la definición del invocador en el contexto de Spring Framework

La forma recomendada de cargar el fichero de Spring es mediante una sentencia *include* en el fichero de servicios.

applicationContext-services.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns=http://www.springframework.org/schema/beans
       xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
       xmlns:p="http://www.springframework.org/schema/p"
       xmlns:context=http://www.springframework.org/schema/context
       xmlns:tx=http://www.springframework.org/schema/tx
       xsi:schemaLocation="
            http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
            http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-2.5.xsd
            http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-2.5.xsd">


    <import resource="applicationContext-OptiwsService.xml"/>


        <!-- ============================================================= -->
        <!--           Definicion de todos los servicios de la aplicacion  -->
        <!-- ============================================================= -->

[...]

</beans>
```

5.3. Enlazar la dependencia en la fachada e implementar llamada

En el mismo fichero applicationContext-services.xml del apartado anterior están las definiciones de las fachadas del proyecto. Debe añadirse la dependencia a la fachada como se haría con cualquier otro servicio del proyecto.

Una vez añadida la dependencia a la fachada, podrán implementarse métodos de uso en esta que utilicen el

invocador creado.

applicationContext-services.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns=http://www.springframework.org/schema/beans
       xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
       xmlns:p="http://www.springframework.org/schema/p"
       xmlns:context=http://www.springframework.org/schema/context
       xmlns:tx=http://www.springframework.org/schema/tx
       xsi:schemaLocation="...">


    <import resource="applicationContext-OptiwsService.xml"/>


        <!-- ============================================================== -->
        <!--            Definicion de todos los servicios de la aplicacion  -->
        <!-- ============================================================== -->
      <bean id="facade"
class="atlasfrm.samples.services.facade.BloquefuncionalnFacadeImpl"
            p:clienteService-ref="clienteService"
            p:optiService-ref="OptiwsService"
      />

[...]

</beans>
```

BloquefuncionalnFacadeImpl.java

```
@Service
public class BloquefuncionalnFacadeImpl implements BloquefuncionalnFacade {

    private OptiwsService optiService;

    public void setOptiService(OptiwsService optiService) {
        this.optiService = optiService;
    }


    /**
     * Llamada al WS de Opti
     * @return numero generado por el webservice
     */
    public String getNumero() throws ServiceException {

        try {
            return this.optiService.codigoDeclaracion2("40", "10", "1");
        } catch (RemoteException e) {
            log.error("Error en llamada al WS.", e);
            throw new ServiceException("Error obteniendo código", e);
        }

    }

}
```

6. ENLACES RELACIONADOS

Producto	URL
Axis2	http://ws.apache.org/axis2/
Documentación online del plugin axis2-wsdl2code-maven-plugin	http://ws.apache.org/axis2/tools/1_4/maven-plugins/maven-wsdl2code-plugin.html